

# FLARE: Finetuning ReLU With FIRE for Efficient Long-Context Inference

Michael Moffatt<sup>§\*</sup>, Junyi Luo<sup>†\*</sup>, Haoran Cheng<sup>§</sup>, Qilong Wang<sup>§</sup>, Xinting Jiang<sup>†</sup>,  
Guanchen Tao<sup>§</sup>, Shiwei Liu<sup>§</sup>, Kauna Lei<sup>†</sup>, Gregory Kielian<sup>†</sup>, Mehdi Saligane<sup>†</sup>

<sup>§</sup>University of Michigan, Ann Arbor, MI, USA

<sup>†</sup>Brown University, Providence, RI, USA

<sup>‡</sup>Google Research, Mountain View, CA, USA

\*Equally Credited Authors (ECAs)

**Abstract**—Deploying large language models (LLMs) on resource-constrained edge devices, such as mobile phones or IoT devices, is highly desirable for enabling secure, personalized on-device AI. However, there are significant challenges due to these models’ high computational and memory demands. A key bottleneck lies in the Transformer’s attention block, especially when handling long contexts. Techniques like model architectures with Rectified Linear Unit (ReLU) activations for Softmax and FIRE positional encoding (a resource-efficient, automatic context-length-scaling alternative to Rotary Positional Embedding (RoPE)) have each independently shown promise in reducing the computational complexity of the attention block, but the proper alchemy for combining their benefits remains underexplored. In this paper, we show a method for combining FIRE and ReLU that maintains low-validation loss at long contexts. We also introduce FLARE, a new algorithm that further improves efficiency by removing operations from the learned relative position encoding in FIRE. Our approach leads to faster inference on long sequences, robust generalization to varying context lengths, and lower validation loss compared to baseline models. FLARE achieves a significant reduction in power and area consumption. On custom hardware, it achieves a 6× higher operating frequency than Softmax, while occupying 57× less silicon area (measured under different throughput settings) and consuming 600× less energy. Our results indicate that FLARE represents a significant step towards deploying powerful LLMs efficiently on resource-limited devices. Code and hardware designs are publicly available at: <https://github.com/ReaLLMASIC/nanoGPT>.

**Index Terms**—Softmax, ReLU, LLM, Inference

## I. INTRODUCTION

Transformers [1] have become the backbone of modern natural language processing due to their scalability and performance, enabled by self-attention mechanisms. However, deploying these models efficiently on edge devices remains a challenge, particularly for long-context applications. Such devices typically lack the computational resources and memory bandwidth to implement advanced latency-reduction techniques for Softmax attention—such as FlashAttention [2], [3]—creating bottlenecks that limit deployment.

This has led to growing interest in alternatives to Softmax that are more hardware-efficient. ReLU-based attention is one such approach, offering significant computational advantages through simpler, element-wise operations. Yet, it is well documented that training from scratch with ReLU-attention results in degraded performance compared to Softmax ([4], [5], [6]). The potential efficiency improvements despite the performance gap has prompted numerous attempts to introduce additional operations

and normalization techniques to mitigate the shortcomings of ReLU, including following the PV multiplication with a layernorm [6] as well as specialized and learned divisors for the result of the ReLU activation ([4], [5]).

However, prior work has largely overlooked a key opportunity: instead of training ReLU attention from scratch, can we fine-tune it into already pre-trained Softmax-based Transformers? This strategy has the potential to retain the high performance of Softmax pre-training while leveraging the inference efficiency of ReLU.

In parallel, recent advances in positional encoding have improved long-context generalization. While rotary position encodings (RoPE) have become a standard for extending context length ([7], [8]), the operation involves pairwise rotations that are not optimal for hardware acceleration. FIRE [9], an additive and dynamically-computable alternative, offers greater hardware simplicity while enabling strong performance in long-context settings.

Our motivation stems from unifying these two strengths—ReLU’s efficiency and FIRE’s scalability—to achieve state-of-the-art performance and efficiency at long context lengths. By combining them in a fine-tuning pipeline, we aim to demonstrate that pre-trained Transformer models can be adapted post-hoc for long-context, low-power inference, without requiring retraining from scratch.

To our knowledge, this is the first work that:

Investigates fine-tuning element-wise activations (e.g., ReLU) in place of Softmax in already pre-trained models.

Analyzes how input/output activation distributions shift during this transition, similar in spirit to fine-tuning normalization layers [10], [11].

Demonstrates that networks can adapt to asymmetric and unbounded activation functions through fine-tuning, challenging the assumption that attention activations must preserve Softmax-like symmetry or boundedness.

**Our contributions are as follows:**

- Demonstrate that pre-training with Softmax and fine-tuning to ReLU outperforms training with ReLU alone under the same token budget.
- Propose a fine-tuning strategy that simultaneously enables long-context generalization and memory efficiency.

- Introduce the FLARE algorithm, which reorders FIRE and ReLU operations at inference to eliminate addition operations in FIRE, boosting compute efficiency.
- Evaluate our design in a hardware implementation and provide a performance, power, and area (PPA) analysis that demonstrates significant gains ( $57\times$  less silicon area  $600\times$  less energy) from ReLU over Softmax on edge accelerators.
- Release a CUDA kernel and PyTorch code base for accelerated ReLU-attention forward passes and fine-tuning of FLARE models.

## II. BACKGROUND

Custom hardware for Transformer models must implement specific self-attention mechanisms and positional encoding strategies at a low level. These choices significantly impact efficiency, as they determine the types of operations involved and how well those operations can be parallelized.

### A. Softmax Inefficiency in Hardware

The numerically-safe Softmax function is defined as:

$$\text{Softmax}(x_i) = \frac{e^{x_i - x_{\max}}}{\sum_j e^{x_j - x_{\max}}} \quad (1)$$

where  $x_{\max} = \max x_j$  is the maximum value per row of  $QK^T$  results. Implementing Softmax efficiently in hardware poses significant challenges:

- 1) **Complex Arithmetic in Implementation:** Implementing the exponential function in hardware requires approximations or lookup tables, adding need for larger silicon area.
- 2) **Data Dependencies:** The amount of maximum and sum computations increase with context length, introducing delays before subsequent steps. This inhibits efficient pipelining, and reduces achievable parallelism.
- 3) **Long-Context Inefficiencies:** Challenges in pipelining and the need to maintain precision for accumulated values – especially for the denominator – will inflate the size required for this module, where complexity scales with the maximum supported context length.

### B. Softmax Variants

Many works have proposed variants to Softmax in attention. [12] introduced Sparsemax, which is designed to generate sparse probability distributions and allow for outputs to be set to zero. [13] built upon this with  $\alpha$ -entmax, a generalization that bridges Softmax and Sparsemax, combining the sparsity of Sparsemax with smoother, differentiable curvature, unlike the piecewise linear nature of Sparsemax. Finally, [14] argues that the effectiveness of Softmax is due to its implicit regularization of the Frobenius norm of the attention matrix, which ensures that backpropagation doesn't lead to too large gradients. They propose a polynomial function that regularizes the Frobenius norm and achieves better performance on vision tasks. We decided to pursue ReLU for its hardware and computational efficiency compared to other methods.

### C. ReLU as a Softmax Replacement

The Rectified Linear Unit (ReLU) function is defined as:

$$\text{ReLU}(x_i) = \max(x_i, 0) \quad (2)$$

Replacing Softmax with ReLU offers significant advantages for edge hardware:

- 1) **Simplified Arithmetic:** No exponentiation, division, multiplication, or addition is required.
- 2) **Element-wise Operations:** ReLU can be applied independently to each element, allowing full parallelization and simplifying pipeline implementations.
- 3) **Long-Context Efficiency:** Via introducing full parallelism, removing the normalization terms, and simplification of the non-linearity, the total size of the resulting implementation of ReLU is minimized and independent of the model's max context length.

### D. Long-Context Scaling of Positional Embeddings

A key flaw with Absolute Position Embeddings is the inability to achieve extrapolation at inference time for inputs longer than seen at training. RoPE [7] and Learned Absolute Position Embeddings [1] both tie a trained model to a specific maximum supported context length. Should a longer context length be desired, the Learned Absolute Position Embeddings table will become larger and linearly grow in number of parameters with the maximum context length. RoPE can be extended to support larger context lengths, but requires fine-tuning, e.g. via interpolation [8], at the longer context length before it can be used.

Functional Interpolation for Relative Position Encoding (FIRE) [9] is a relative positional encoding method that results in additive bias after the  $QK^T$  multiplication. Like many relative embeddings, this removes explicit algorithmic dependencies on the max context length. Moreover, compared alongside other relative position embeddings, FIRE has SOTA-level length generalizability, namely, one can train for high accuracy on a smaller context-length and transfer this high accuracy to several times the context length seen by the model during training. The reliance on addition vs RoPE allows for smaller area and the scaling method (larger element-wise additive bias matrix) prevents need for increasing precision of individual blocks when scaling hardware implementations.

FIRE is not the only technique proposed for long context extrapolation. [15] proposed AliBi, a fixed and non-learned bias on the query-key matrix to reduce scores for query-key pairs that are farther apart. By creating a bias dependent on relative distances, AliBi demonstrates strong length generalization, being able to extrapolate to a length of 3072 when trained on 512 context length. [16] improves on AliBi's performance by instead building on RoPE and adding an exponential decay term to the rotation matrix of embeddings. This model, LeX, shows better performance on shorter and longer texts than the trained context length. Similarly, KERPLE [17] modified relative position encodings by leveraging kernel functions to efficiently compute positional biases. KERPLE similarly shows length generalization improvements over AliBi.

Compared to these methods, FIRE demonstrates robust length generalization capabilities. [9] reports that FIRE outperforms both AliBi and KERPLE across several benchmarks for long contexts. Finally, some work has shown that FIRE can be combined with position specific embeddings for even greater context length generalizability [18].

### III. METHODS

In this section, we describe the methodologies employed in our study, including the experimental setup, datasets, training configurations, the detailed recipes for integrating ReLU and FIRE into Transformer models, the implementation of the FLARE algorithm, the development of the ReLU-augmented FlashAttention mechanism, and the hardware implementation and performance, power, and area analysis.

#### A. Implementation of the FLARE Algorithm

To further improve computational efficiency, we proposed a fused FLARE algorithm which combines FIRE position encoding with sparsity producing ReLU. This algorithm reorders the comparison of ReLU to occur before the FIRE bias addition step for positional encoding. In the standard attention mechanism with FIRE and ReLU, the attention score matrix  $\mathbf{S}$  is computed as:

$$s_{ij} = \max(a_{ij} + f_{ij}, 0) \quad (3)$$

where  $\mathbf{A} = \mathbf{QK}^\top$  is the scaled dot-product of queries and keys, and  $\mathbf{F}$  represents the FIRE bias matrix.

The FLARE algorithm optimizes this computation by observing that if  $f_{ij} \leq -a_{ij}$ , then  $s_{ij} = 0$ , and the addition can be skipped. This leads to a case-wise expression:

$$s_{ij} = \begin{cases} 0, & \text{if } f_{ij} \leq -a_{ij} \\ a_{ij} + f_{ij}, & \text{otherwise} \end{cases} \quad (4)$$

By implementing this logic, we can avoid unnecessary addition operations when  $f_{ij} \leq -a_{ij}$ .

We conducted experiments to measure the proportion of operations which take this branch. From evaluation of the final checkpoint inference, we determined that the "Probability Matrix" output of ReLU for all layers is on average 98.9% zeros in the lower-left triangle for causal attention. This high degree of sparsity means that the condition  $f_{ij} \leq -a_{ij}$  holds 98.9% of the time, and the comparison alone will suffice for 98.9% of ReLU outputs.

This observed sparsity is consistent with findings from ([6]), which shows that "null attention"—rows of all-zero attention scores—emerges naturally in ReLU-based attention. As the model trains, it learns to suppress irrelevant or distant tokens by producing negative scores, which ReLU clips to zero.

Fig. 1 illustrates the original attention architecture, the ReLU with FIRE, and the optimized FLARE module.

When training or fine-tuning models, Flash-Attention [2] greatly accelerates the Softmax's calculation in attention and reduces the attention memory footprint – allowing longer contexts to be trained on GPUs with smaller VRAMs. In order to facilitate adoption of fine-tuning of FLARE into pre-trained

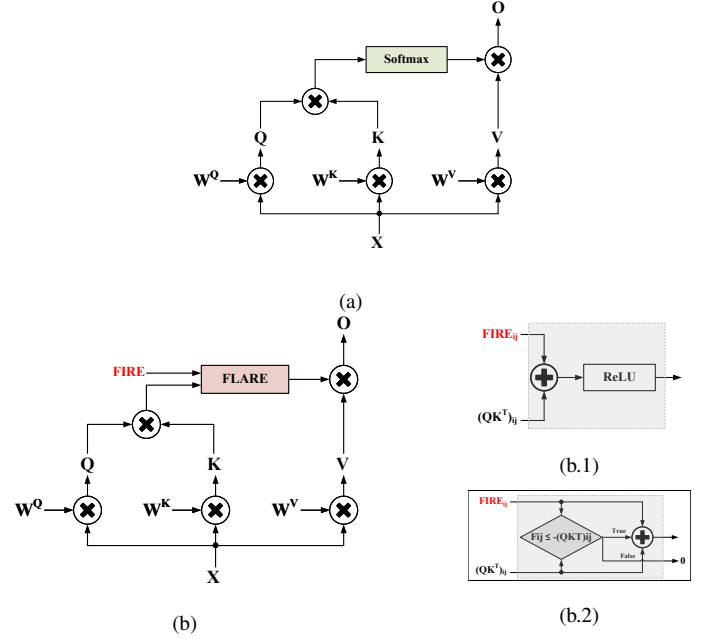


Fig. 1. (a) Structure of original attention architecture. (b) Overall structure of FLARE-integrated attention architecture, with the gray-highlighted section (FLARE module) zoomed in on the right to illustrate two different hardware implementations: (b.1) Original design with FIRE and ReLU, and (b.2) Optimized design of the FLARE algorithm.

models for increased hardware efficiency, we contribute a custom CUDA Kernel.

**Development of Custom CUDA Kernels:** We developed custom CUDA kernels that implements a FLARE-based attention mechanism for the max portion of a  $\max(FIRE_{ij} + QK^T, 0)$  FLARE implementation. This utilizes tiling optimization techniques inspired by FlashAttention [2]. By leveraging shared memory and tiling strategies, we optimized memory access patterns and reduced bandwidth requirements. This is crucial for handling large attention matrices, especially with long sequences. Furthermore, the design of our Flash-FLARE-Attention kernels is modular and compatible with the architectural improvements introduced in FlashAttention2 [3].

The simplified Flash-FLARE-Attention forward pass algorithm is outlined in Algorithm 1.

#### Algorithm 1 FlashFlareAttention

---

**Require:** Matrices  $Q, K, V \in \mathbb{R}^{N \times d}$  in HBM, on-chip SRAM of size  $M$ .

- 1: Set block sizes  $B_c = \lfloor \frac{M}{d} \rfloor$ ,  $B_r = \min(\lfloor \frac{M}{d} \rfloor, d)$ .
- 2: Initialize  $O = (0)^{N \times d}$ , in HBM.
- 3: Divide  $Q$  into  $T_r = \lceil \frac{N}{B_r} \rceil$  blocks  $Q_1, \dots, Q_{T_r}$  of size  $B_r \times d$  each, and divide  $K, V$  into  $T_c = \lceil \frac{N}{B_c} \rceil$  blocks  $K_1, \dots, K_{T_c}$  and  $V_1, \dots, V_{T_c}$ , of size  $B_c \times d$  each.
- 4: Divide  $O$  into  $T_r$  blocks  $O_1, \dots, O_{T_r}$ , of size  $B_r \times d$  each
- 5: **for**  $j = 1$  **to**  $T_c$  **do**
- 6:   Load  $K_j, V_j$  from HBM to SRAM.
- 7:   **for**  $i = 1$  **to**  $T_r$  **do**
- 8:     Load  $Q_i, O_i$  from HBM to SRAM.
- 9:     On chip, compute  $S_{ij} = Q_i K_j^T \in \mathbb{R}^{B_r \times B_c}$  (element-wise).
- 10:     On chip,  $\hat{P}_{ij} = \max(S_{ij}, 0)$  (element-wise).
- 11:     Write  $O_i \leftarrow \hat{P}_{ij} V_j$  to HBM.
- 12:   **end for**
- 13: **end for**
- 14: **return**  $O$ .

---

## B. Dataset and Training Settings

We utilize the OpenWebText dataset [19], an open-source replication of the WebText corpus used for training GPT-2. The dataset consists of high-quality web content extracted from Reddit submissions with an upvote score of at least 3, providing a diverse and representative sample of internet text.

For model training, we use a maximum context length of 1024 tokens. Unless otherwise specified, we initialize our models from the GPT-2 small checkpoint, which contains approximately 124 million parameters. Training is performed using a batch size of 32 and a learning rate of  $6 \times 10^{-4}$ , optimized with the Adam optimizer [20] and a cosine learning rate decay schedule. Gradient clipping is applied with a maximum norm of 1.0 to stabilize training.

All experiments are performed on NVIDIA A100 GPUs, leveraging their computational capabilities for efficient training and inference of large language models.

## C. Recipes for ReLU Integration

We conducted a series of experiments to compare different training recipes for effectively integrating the ReLU activation function as a replacement for the Softmax function in the attention mechanism of Transformer models. Our goal was to identify strategies that mitigate the performance degradation typically observed when training models with ReLU from scratch.

1) *Experiment 1: Training with ReLU from Scratch vs. Fine-tuning with ReLU:* In the first experiment (Section IV-A), over the same total number of training iterations, we compared the performance of models trained with ReLU from scratch against models that were first trained with Softmax and then fine-tuned with ReLU.

- **Training with ReLU from Scratch:** We trained a Transformer model with the ReLU-based attention mechanism from scratch for 30,000 iterations. The model was initialized randomly and did not utilize any pre-trained weights. We tracked the validation loss at regular intervals to assess the learning progress.
- **Fine-tuning ReLU into a Softmax Pre-trained Model:** We first trained the model with the standard Softmax-based attention mechanism for 20,000 iterations. After this initial training phase, we replaced the Softmax function with ReLU and fine-tuned the model for an additional 10,000 iterations. We monitored the validation loss throughout both phases.

2) *Experiment 2: Sequential vs. Simultaneous Integration of ReLU and FIRE:* In the second experiment (Section IV-B), we investigated the impact of integrating ReLU and FIRE positional embeddings into a pre-trained model, comparing sequential integration in different orders with simultaneous integration. Our goal was to determine the most effective strategy for incorporating both ReLU and FIRE and to understand how the order of integration affects model performance and training stability.

- **Simultaneous Training of ReLU and FIRE:** Starting from the GPT-2 124M checkpoint, we simultaneously

replaced Softmax with ReLU in the attention mechanism and the absolute positional embeddings with FIRE. We then integrated both changes at once and trained the model for 20,000 iterations.

- **Sequential Training with ReLU Followed by FIRE:** We first fine-tuned the GPT-2 124M model by replacing Softmax with ReLU while retaining absolute positional embeddings. We trained the model for 10,000 iterations to allow it to adapt to the ReLU activation in the attention mechanism. Then, we replaced the absolute positional embeddings with FIRE and continued fine-tuning for another 10,000 iterations.
- **Sequential Training with FIRE Followed by ReLU:** Conversely, we also explored the effect of introducing FIRE before ReLU. We replaced the absolute positional embeddings with FIRE while retaining Softmax and trained the model for 10,000 iterations to allow it to adapt to FIRE. Then, we replaced Softmax with ReLU and fine-tuned the model for an additional 10,000 iterations.

Throughout the training process, we tracked the validation loss and monitored the input and output distributions of the ReLU activation function to observe how the model adapted to these changes.

By comparing these three strategies, we aimed to determine the most effective approach for incorporating both ReLU and FIRE into a pre-trained Transformer model. This comparison allowed us to assess how the order of integration influences the model’s adaptability, training stability, and final performance in terms of validation loss and length generalization capabilities.

3) *Experiment 3: Length Generalization Evaluation:* We assessed the ability of each trained model to generalize to longer context lengths than those used during training (Section IV-B).

- **Evaluation on Extended Context Lengths:** We evaluated all models on context lengths of 2048 and 4096 tokens, to test length generalization from their trained maximum context length of 1024 tokens. We measured the validation loss at these extended lengths to determine how well each training recipe enabled length generalization.

This evaluation allowed us to understand the impact of different integration strategies on the models’ ability to handle longer sequences, and acts as a figure of merit for how well FIRE was integrated.

4) *Experiment 4: Benchmarking Results on Larger Models:* In our final experiment, we verified our experimental results on larger-scale models by evaluating Qwen2 0.5B and 1.5B parameter variants across four attention configurations: (1) standard Softmax, (2) ReLU attention, (3) FIRE positional encoding with Softmax, and (4) FLARE, which combines ReLU attention with FIRE. For each configuration, we measured validation loss after 10k training iterations on openwebtext, accuracy on HellaSwag (acc\_norm), and both inference and training throughput (in tokens/sec).

## D. Hardware Implementation and PPA Analysis

We contribute metrics for hardware efficiency gains of using the ReLU algorithm over Softmax in custom hardware for

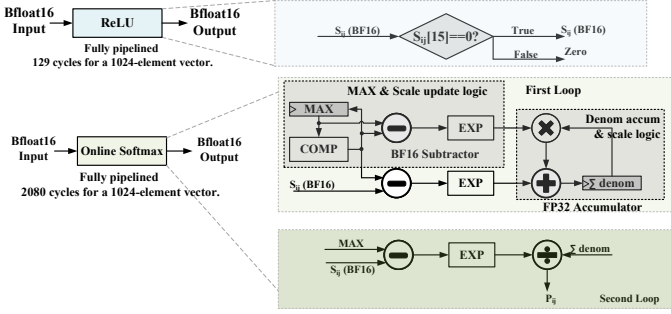


Fig. 2. Hardware implementation of ReLU and Softmax.

FLARE implementation, as shown in Fig. 2. We implement these algorithms in custom hardware, performing performance, power, and area (PPA) analysis. For comparison, the baseline Softmax hardware implementation employs the online-normalization algorithm [21], the same technique used in several other Softmax designs [22] [23]. To ensure a fair comparison, we exclude the SRAM buffer from the PPA analysis. Although the online Softmax implementation inherently requires this buffer for its two sequential loops computation, the same SRAM can be shared across other accelerator stages and thus should not be uniquely attributed to Softmax.

1) *Hardware implementations:* We synthesized and performed automatic place-and-route (APR) for hardware implementations of *ReLU* and *Softmax* using the 16 nm technology node. Since a single ReLU module is too small to allow Fusion Compiler to fully exploit optimization, we increased the ReLU bus throughput to 8 per cycle to obtain more representative results. The online Softmax implementation remains unchanged. All experiments use the 16 nm process at the 0.8 V typical-typical (TT) corner and 85 °C. For a fair comparison, identical input/output delay constraints are applied to both the online Softmax and ReLU designs.

2) *Performance Metrics:* We evaluated ReLU and Softmax across four key metrics: clock frequency (MHz), silicon area ( $\mu\text{m}^2$ ), power ( $\mu\text{W}$ ), and energy (nJ).

3) *Analysis and Comparison:* By comparing the synthesized hardware modules, we quantified the efficiency gains of ReLU over Softmax in terms of speed, power efficiency, and hardware resource requirements. The results are detailed in Section V-B.

## IV. RESULTS

### A. Pre-training with Softmax vs ReLU

We start by comparing the performance of ReLU fine-tuning vs training from scratch.

Fig. 3 shows that, over the same number of total training iterations, a model trained with Softmax and fine-tuned with ReLU performs better than a model trained using just ReLU. After replacing Softmax with ReLU, the validation loss quickly overtakes that of the model with ReLU from scratch.

Furthermore, due to the benefits of highly optimized Softmax attention on GPU with FlashAttention, we observed an all-in-all reduction in training time for the fine-tuning recipe.

Validation Loss Comparison: Training ReLU from Scratch vs Finetuning in ReLU

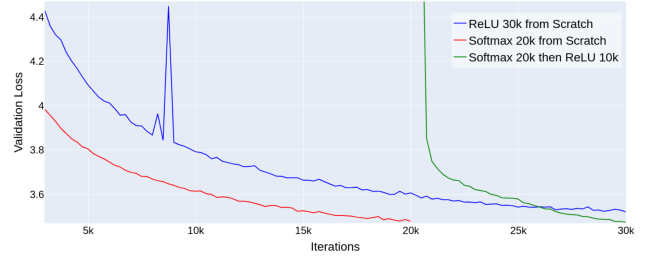


Fig. 3. Comparison of validation loss between ReLU trained for 30k iterations vs fine-tuning for 10k iterations after training Softmax for 20k.

Validation Loss Comparison: FIRE Fine-tuning

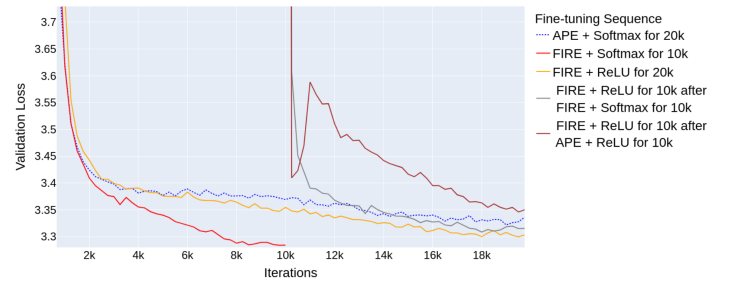


Fig. 4. Validation loss for Softmax variants fine-tuned with FIRE.

Inference Validation Losses across Context Lengths

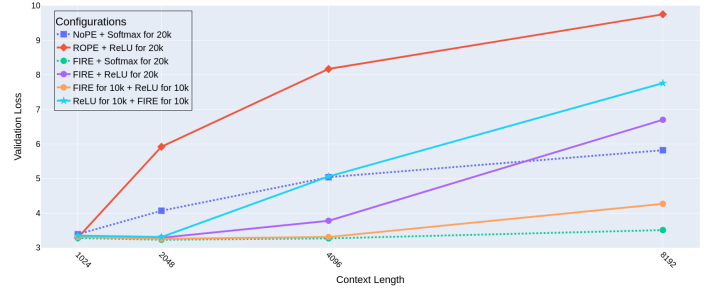


Fig. 5. Validation loss measurements for each fine-tuning recipe, with NopE and RoPE provided as baselines. Note how fine-tuning FIRE for 10k iterations then ReLU 10k iterations yields strong length-generalization vs other FIRE + ReLU recipes.

Training on a single A100 80GB GPU, we found the training from scratch with ReLU took 59% longer than the Softmax-then-ReLU fine-tuning recipe. ReLU pre-training took 17 hours, where as the total training time from Softmax and ReLU fine-tuned took 12 hours in total.

### B. Results of Different Recipes for Fine-tuning ReLU and FIRE into Models

Between fine-tuning simultaneously with ReLU and FIRE, fine-tuning sequentially (first FIRE then ReLU or first ReLU then FIRE), Fig. 4 shows that the best validation losses are nearly the same with fine-tuning simultaneously and fine-tuning FIRE-then-ReLU.

However, we find that only fine-tuning with FIRE first then ReLU was able to successfully impart length-generalization capabilities to the model, as evidenced in Fig. 5.

TABLE I  
QWEN2 ATTENTION VARIANTS: VALIDATION LOSS, ACCURACY, AND SPEED  
(AFTER 10K FINE-TUNING ITERATIONS)

Model	Type	Val. Loss	Hellaswag	Train tok/s	Infer tok/s
Qwen2 0.5B	Softmax	3.493	0.313	5258	30.7
	ReLU	3.716	0.315	7056	41.8
	FIRE	3.462	0.333	5497	31.9
	FLARE	3.710	0.324	7065	42.7
Qwen2 1.5B	Softmax	3.378	0.334	4048	26.2
	ReLU	3.379	0.340	5044	33.9
	FIRE	3.341	0.351	4149	26.4
	FLARE	3.376	0.335	5176	34.7

Finally, we found our results scale to larger models. On Qwen2 0.5B and 1.5B, as shown in Table I, we observed that the FLARE variant maintained comparable validation loss and HellaSwag accuracy to the other configurations, while delivering superior inference and training efficiency.

### C. Input and Output Statistics

We monitored input and output statistics during fine-tuning to understand how the model adapts its input and output ranges when accommodating a vastly different activation function in the attention module.

Fig. 6a and Fig. 6b show that the output distributions for ReLU stabilize after around 3k iterations. While Softmax’s distribution of inputs was very symmetrical, ReLU’s distribution pulled towards the negative side, clearly selecting a smaller proportion of its outputs – on average only 1.1% of its outputs – for the non-zero values in the output probability matrix.

## V. PROFILING AND HARDWARE IMPLEMENTATION

### A. Performance Evaluation

We compared the forward pass inference time between Flash-FLARE-Attention and FlashAttention across different context lengths. As shown in Fig. 7, Flash-FLARE-Attention achieves an average speed up of 3.8x over FlashAttention for context lengths of 512 to 4096.

### B. Hardware Comparison

We conducted a PPA analysis between ReLU and Softmax implementations synthesized in the 16nm node. As shown in Fig. 2, both ReLU and Softmax engines use the **BF16** format on their inputs and outputs for a fair evaluation. For a vector of length 1024:

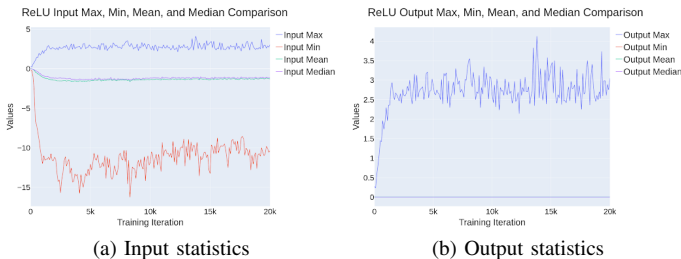


Fig. 6. Input and output statistics of ReLU fine-tuned from a Softmax pre-trained model over 20k iterations.

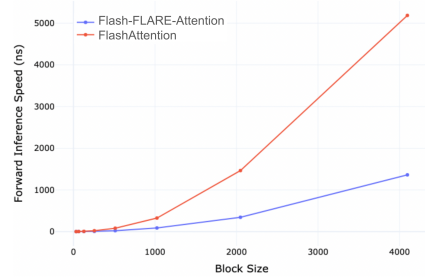


Fig. 7. Inference speed comparison between Flash-FLARE-Attention and FlashAttention.

TABLE II  
HARDWARE COMPARISON OF ReLU AND Softmax

Metric	ReLU	Softmax
Frequency (MHz)	3226	515
Area ( $\mu\text{m}^2$ )	136.08	7838.21
Power ( $\mu\text{W}$ )	80.7	509.2
Energy (nJ)	0.0032	2.056
Latency (ns)	39.98	4038.8

- **ReLU** finishes in 129 cycles. Because determining the output requires only the sign bit, the datapath is fully parallel with a bus width of 8.
- **Softmax** takes 2080 cycles. 1035 clock cycles for the first loop and 1045 clock cycles for the second loop.

As summarized by Table II, ReLU has a  $6\times$  frequency increase over Softmax. The ReLU area is 1.7% the size of the silicon implementation of Softmax. ReLU uses 0.156% the energy compared to Softmax.

## VI. CONCLUSION

In this paper, we explore fine-tuning as a method for improving the performance of ReLU in the attention block for Transformers. We show that with the correct fine-tuning recipe, one can introduce FIRE and ReLU into a pre-trained model, obtaining the length-generalization and improving final validation loss. Our proposed FLARE algorithm efficiently fuses ReLU and FIRE, reducing computational complexity via reordering and merging their operations.

Flash-FLARE-Attention is proposed to alleviate the hardware barrier for fine-tuning FLARE, enabling more hardware-efficient models.

Finally, our analysis shows substantial performance, power, and area advantages over Softmax, making it highly suitable for edge devices.

## REFERENCES

for quantized transformers,” in *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*. IEEE, 2023, pp. 1–6.

- [1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, E. Kaiser, and I. Polosukhin, “Attention is all you need,” in *Advances in Neural Information Processing Systems*, 2017, p. 6000–6010.
- [2] T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré, “Flashattention: Fast and memory-efficient exact attention with io-awareness,” *Advances in Neural Information Processing Systems*, 2022.
- [3] T. Dao, “Flashattention-2: Faster attention with better parallelism and work partitioning,” in *International Conference on Learning Representations (ICLR)*, 2023, arXiv:2307.08691. [Online]. Available: <https://arxiv.org/abs/2307.08691>
- [4] M. Wortsman, J. Lee, J. Gilmer, and S. Kornblith, “Replacing softmax with relu in vision transformers,” *arXiv preprint arXiv:2309.08586*, 2023.
- [5] K. Shen, J. Guo, X. Tan, S. Tang, R. Wang, and J. Bian, “A study on relu and softmax in transformer,” *arXiv preprint arXiv:2302.06461*, 2023.
- [6] B. Zhang, I. Titov, and R. Sennrich, “Sparse attention with linear units,” *arXiv preprint arXiv: 2104.07012*, 2021.
- [7] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, “Roformer: Enhanced transformer with rotary position embedding,” 2023. [Online]. Available: <https://arxiv.org/abs/2104.09864>
- [8] S. Chen, S. Wong, L. Chen, and Y. Tian, “Extending context window of large language models via positional interpolation,” 2023. [Online]. Available: <https://arxiv.org/abs/2306.15595>
- [9] S. Li, C. You, G. Guruganesh, J. Ainslie, S. Ontanon, M. Zaheer, S. Sanghai, Y. Yang, S. Kumar, and S. Bhojanapalli, “Functional interpolation for relative positions improves long context transformers,” in *The Twelfth International Conference on Learning Representations*, 2024.
- [10] S. Heimersheim, “You can remove gpt2’s layernorm by fine-tuning,” *arXiv preprint arXiv:2409.13710*, 2024. [Online]. Available: <https://arxiv.org/abs/2409.13710>
- [11] J. Zhu, X. Chen, K. He, Y. LeCun, and Z. Liu, “Transformers without normalization,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025, arXiv:2503.10622. [Online]. Available: <https://arxiv.org/abs/2503.10622>
- [12] A. F. Martins and R. Astudillo, “From softmax to sparsemax: A sparse model of attention and multi-label classification,” *arXiv preprint arXiv:1602.02068*, 2016. [Online]. Available: <https://arxiv.org/pdf/1602.02068>
- [13] B. Peters, V. Niculae, and A. F. Martins, “Sparse sequence-to-sequence models,” *arXiv preprint arXiv:1905.05702*, 2019. [Online]. Available: <https://arxiv.org/pdf/1905.05702>
- [14] S. Hemanth, J. Zheng, Y. Ji, Z. Wenbo, and S. Lucey, “Rethinking softmax: Self-attention with polynomial activations,” *arXiv preprint arXiv:2410.18613*, 2024. [Online]. Available: <https://arxiv.org/pdf/2410.18613>
- [15] O. Press, N. A. Smith, and M. Lewis, “Attention with linear biases (alibi),” *arXiv preprint arXiv:2108.12409*, 2022. [Online]. Available: <https://arxiv.org/pdf/2108.12409>
- [16] Y. Sun, L. Dong, B. Patra, S. Ma, S. Huang, A. Benhaim, V. Chaudhary, X. Song, and F. Wei, “A length-extrapolatable transformer,” *arXiv preprint arXiv:2212.10554*, 2022. [Online]. Available: <https://arxiv.org/pdf/2212.10554>
- [17] T.-C. Chi, T.-H. Fan, P. J. Ramadge, and A. I. Rudnicky, “Kerple: Kernelized relative positional embedding for length extrapolation,” *arXiv preprint arXiv:2205.09921*, 2022. [Online]. Available: <https://arxiv.org/pdf/2205.09921>
- [18] S. McLeish, A. Bansal, A. Stein, N. Jain, J. Kirchenbauer, B. R. Bartoldson, B. Kailkhura, A. Bhatele, J. Geiping, A. Schwarzschild, and T. Goldstein, “Transformers can do arithmetic with the right embeddings,” *arXiv preprint arXiv:2405.17399*, May 2024. [Online]. Available: <https://arxiv.org/abs/2405.17399>
- [19] A. Gokaslan and V. Cohen, “Openwebtext corpus,” <http://Skylion007.github.io/OpenWebTextCorpus>, 2019.
- [20] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014. [Online]. Available: <https://arxiv.org/abs/1412.6980>
- [21] M. Milakov and N. Gimelshein, “Online normalizer calculation for softmax,” *arXiv preprint arXiv:1805.02867*, 2018.
- [22] J. R. Stevens, R. Venkatesan, S. Dai, B. Khailany, and A. Raghunathan, “Softermax: Hardware/software co-design of an efficient softmax for transformers,” in *Proceedings of the 58th ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 469–474.
- [23] G. Islamoglu, M. Scherer, G. Paulin, T. Fischer, V. J. Jung, A. Garofalo, and L. Benini, “Ita: An energy-efficient attention and softmax accelerator